FeinCMS Articles Documentation

Release 0.3

Incuna Ltd

February 06, 2017

Contents

This is an extensible FeinCMS content article system for Django, designed to provide a simple Article model that is extensible.

What is an Article? Many things! You can use it as a news section, as a knowledge base, as a catalogue of pdfs, or pretty much anything else you can make it fit with.

CHAPTER 1

Installation and setup

Firstly, get the package:

```
pip install feincms-articles
```

You will then need to add `articles` to your `INSTALLED_APPS` setting:

```
INSTALLED_APPS = (
    # ...
    'articles',
)
```

Before proceeding with `manage.py syncdb`, you will need to create some content types and you may want to add some article extensions. By default the articles module has a basic set of content fields such as title, summary and content.

CHAPTER 2

Registering ContentTypes

The `Article` model extends from `the FeinCMS Base model`, so you need to create some FeinCMS content types to add to your Articles. No types are created by default, because there is no way to unregister them. A sane default might be to create `MediaFileContent` and `RichTextContent` models; you can do this by adding the following lines somewhere into your project, for example at the bottom of a `models.py` file that will be processed anyway:

```
from feincms.content.richtext.models import RichTextContent
from feincms.content.medialibrary import MediaFileContent

from articles.models import Article

Article.register_regions(('top', _('Top content')), ('main', _('Main region')),)

Article.create_content_type(RichTextContent)
Article.create_content_type(MediaFileContent, POSITION_CHOICES=(('block', _('block')), ('left', _('le
```

CHAPTER 3

Extensions

Extensions are a way to add often-used functionality the Article model. The extensions are standard python modules with a `register()` method which will be called upon registering the extension. The `register()` method receives the `Article` class itself and the model admin class `ArticleAdmin` as arguments.

The extensions can be activated by adding the following to a the bottom of a `models.py` file that will be processed anyway:

```
from articles.models import Article

Article.register_extensions(
    'articles.modules.category.extensions.category',
    'feincms.module.extensions.datepublisher',
    'articles.extensions.tags',
    'articles.extensions.thumbnail',
)
```

Articles comes with a number of bundled extensions, or you can use the generic FeinCMS extensions.

CHAPTER 4

Creating your own extensions

To add an extension create a python module that defines a register function that accepts the `Article` class and the `ArticleAdmin` class as arguments and modifies them as required.

Here is the tags extension (similar to articles/extensions/tags.py):

```
def register(cls, admin_cls):
    cls.add_to_class('tags', TaggableManager(verbose_name=_('tags'), blank=True))

    cls.urlpatterns += patterns('taggit.views',
        url(r'^tags/(?P<slug>[^/]+)/$', 'tagged_object_list', {'queryset': cls.objects.active}, name=
    )

    if admin_cls:
        if admin_cls.fieldsets:
            admin_cls.fieldsets[0][1]['fields'].append('tags')
```

CHAPTER 5

Hooking up articles into your application

You have two main options here, depending on your use case. FeinCMS articles can be deployed using Application content, hooking up the URLconf `articles.urls`.

Alternatively you can just use content types to display a list of articles on a page. There is a bundled content type `articles.content.ArticleList` which will render a fixed number of articles.

If you're not using the FeinCMS page module, the urls and views within articles are safe to use without `ApplicationContent`.

The category module also comes with content types for a list of articles belonging to a certain category (`articles.modules.category.content.ArticleCategoryList`) and the list of articles belonging to a set of categories (`articles.modules.category.content.ArticleList`).

There is also a template tag `article_tags.articles`, which will render a list of articles. It takes a optional parameters for `limit` (the number of articles) and the variable to insert the articles list into the context as.

CHAPTER 6

Contents

6.1 Bundled extensions

FeinCMS articles uses the FeinCMS `extensible architecture`, and it comes with a number of built in extensions.

Note: Please note that as of FeinCMS 1.6 you will no longer be able to use the short-form registration

If the extension requires it's own models (like the category extension) then the app containing the models will also need to be added to your `INSTALLED_APPS`.

6.1.1 Location extension

Register: `articles.extensions.location`.

Requires `GeoDjango`.

Attaches a physical location point to each article. We recommend that you set *`ARTICLE_MODELADMIN_CLASS`* to a (subclass of) `django.contrib.gis.admin.OSMGeoAdmin` to get a nicer admin user interface.

6.1.2 Tags extension

Register: `articles.extensions.tags`.

Requires django-taggit.

Adds tagging to articles. Adds views to the article urls at `/tags/<tag>/` to provide a list of articles by tag.

6.1.3 Thumbnail extension

Register: `articles.extensions.thumbnail`.

Adds a simple `ImageField` for use as an article thumbnail. Requires PIL, but you knew that already right?

6.1.4 Category extension

Register: `articles.modules.category.extensions.category`.

Requires the category module to be added to installed apps:

```
INSTALLED_APPS = (
    ...
    'articles.modules.category',
)
```

This is a nested category setup, that is categories can live within other categories. The extension will update the url structure of `articles.urls` to reflect the new structure.

6.2 Available configuration settings

ARTICLE_MODELADMIN_CLASS
Default: `django.contrib.admin.ModelAdmin`

Sets the base class for the `ModelAdmin` used by `Articles`. Note that the class will be monkey patched by the extensions.

6.2.1 Specific to the category extension

CATEGORY_MODELADMIN_CLASS
Default: `django.contrib.admin.ModelAdmin`

Sets the base class for the `ModelAdmin` used by `Category`.

ARTICLE_SHOW_FIRST_CATEGORY
Default: `False`

When set to `True`, this will cause the list of Categories view to redirect to the first Category's list of articles.

ARTICLE_SHOW_DESCENDANTS
Default: `False`

When set to `True`, this will display all articles belonging to any descendant category on the list views.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

a

A

C